Lecture 2

Functions & Modules
Labs Start this Week

- Labs are Wednesday in Upson 142
  - Can do them at home (instructions on web)
  - Bring a laptop if you come to the lab
- Lab is due before beginning of next lab
  - Can check off on the day it is handed out
  - Can check off during consultant hours
- Cannot spend lab time on previous lab
Announcements

If Not Done Already

• Enroll in Piazza
• Sign into CMS
  ▪ Fill out the Survey
  ▪ Complete AI Quiz
• Do AI Quiz by Friday!

Reading (Optional)

• Readings this week
  ▪ Chapter 1 (browse)
  ▪ Chapter 2 (in detail)
  ▪ Chapter 3 (in detail)
• Next week
  ▪ Sections 8.1, 8.2, 8.4, 8.5
Function Calls

• Python supports expressions with math-like functions
  ▪ A function in an expression is a function call
  ▪ Will explain the meaning of this later

• Function expressions have the form \texttt{fun}(x,y,...)

• Examples (math functions that work in Python):
  ▪ \texttt{round}(2.34)
  ▪ \texttt{max(a+3,24)}
Function Calls

- Python supports expressions with math-like functions
  - A function in an expression is a **function call**
  - Will explain the meaning of this later
- Function expressions have the form `fun(x,y,...)`

- **Examples** (math functions that work in Python):
  - `round(2.34)`
  - `max(a+3,24)`
Built-In Functions

• You have seen many functions already
  ▪ Type casting functions: int(), float(), bool()
  ▪ Dynamically type an expression: type()
  ▪ Help function: help()
  ▪ Quit function: quit()

• One of the most import functions is print()
  ▪ print(exp) displays value of exp on screen
  ▪ Will see later why this is important
Built-in Functions vs Modules

- The number of built-in functions is small
  - [http://docs.python.org/3/library/functions.html](http://docs.python.org/3/library/functions.html)

- Missing a lot of functions you would expect
  - **Example**: `cos()`, `sqrt()`

- **Module**: file that contains Python code
  - A way for Python to provide optional functions
  - To access a module, the `import` command
  - Access the functions using module as a *prefix*
Example: Module `math`

```python
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

8/28/17
Example: Module math

>>> import math
>>> math.cos(0)
1.0

>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined

>>> math.pi
3.141592653589793

>>> math.cos(math.pi)
-1.0

To access math functions
Example: Module `math`

```python
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```
Example: Module `math`

```python
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

Functions & Modules

To access math functions

Functions require math prefix!

Module has variables too!
Example: Module math

```python
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0

Other Modules

- io
  - Read/write from files
- random
  - Generate random numbers
  - Can pick any distribution
- string
  - Useful string functions
- sys
  - Information about your OS

Module has variables too!

Functions require math prefix!

To access math functions

8/28/17 Functions & Modules
Reading the Python Documentation

9.2. math — Mathematical functions

This module is always available. It provides access to the mathematical functions defined by the C standard. These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1. Number-theoretic and representation functions

math.ceil(x)
Return the ceiling of x, the smallest integer greater than or equal to x. If x is not a float, delegates to x.__ceil__(), which should return an Integral value.

math.copysign(x, y)
Return a float with the magnitude (absolute value) of x but the sign of y. On platforms that support signed zeros, copysign(1.0, -0.0) returns -1.0.

math.fabs(x)
Return the absolute value of x.

math.factorial(x)
Return x factorial. Raises ValueError if x is not integral or is negative.

math.floor(x)
Return the floor of x, the largest integer less than or equal to x. If x is not a float, delegates to x.__floor__(), which should return an Integral value.

math.fmod(x, y)
Return fmod(x, y), as defined by the platform C library. No C standard is that fmod(x, y) be exactly (mathematically; to omitive precision) equal to x - floor(x/y) * y for some integer n such that the result has the same sign as x and magnitude less than abs(y). Python's x % y returns a result with the sign of y instead, and may not be exactly computable for float arguments. For example, fmod(-1e-100, 1e100) is -1e-100, but the result of Python's -1e-100 % 1e100 is 1e100-1e-100, which cannot be

http://docs.python.org/library
Reading the Python Documentation

http://docs.python.org/library

math.ceil(x)

Return the ceiling of x as a float, the smallest integer value greater than or equal to x.
9.2. math — Mathematical functions

This module is always available. It provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

---

**Function name**

```
math.ceil(x)
```

**Possible arguments**

```
math.ceil(x)
```

**What the function evaluates to**

Return the ceiling of \( x \) as a float, the smallest integer value greater than or equal to \( x \).

---

---

http://docs.python.org/library
Interactive Shell vs. Modules

- Launch in command line
- Type each line separately
- Python executes as you type

Write in a text editor
  - We use Komodo Edit
  - But anything will work
- Load module with import
Using a Module

Module Contents

""" A simple module.

This file shows how modules work
"""

# This is a comment

x = 1+2
x = 3*x
x
Using a Module

Module Contents

""" A simple module.

This file shows how modules work

"""

# This is a comment

x = 1+2

x = 3*x

x
Using a Module

Module Contents

""" A simple module.
This file shows how modules work """

# This is a comment
x = 1+2
x = 3*x
x

Docstring (note the Triple Quotes)
Acts as a multiple-line comment
Useful for code documentation

Single line comment
(not executed)
Using a Module

Module Contents

""""A simple module.

This file shows how modules work
"""

# This is a comment

x = 1+2
x = 3*x
x

---

**Docstring** (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

**Single line comment**
(not executed)

**Commands**
Executed on import
Using a Module

Module Contents

""" A simple module.

This file shows how modules work
"""

```python
# This is a comment

x = 1+2
x = 3*x

# Not a command.
import ignores this
```

---

**Docstring** (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

**Single line comment**
(not executed)

**Commands**
Executed on import
Using a Module

Module Contents

""" A simple module.

This file shows how modules work
"""

# This is a comment

x = 1+2
x = 3*x
x

Python Shell

>>> import module

>>> x
# Using a Module

## Module Contents

""" A simple module.

This file shows how modules work 
"""

# This is a comment

```python
x = 1+2
x = 3*x
x
```

## Python Shell

```python
>>> import module

>>> x

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```
# Using a Module

## Module Contents

""" A simple module.

This file shows how modules work """

```python
# This is a comment
x = 1+2
x = 3*x
x
```

## Python Shell

```python
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> module.x
9
```

"Module data" must be prefixed by module name
Using a Module

<table>
<thead>
<tr>
<th>Module Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;&quot;&quot; A simple module.</td>
</tr>
<tr>
<td>This file shows how modules work &quot;&quot;&quot;&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Python Shell</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt; import module</td>
</tr>
<tr>
<td>&gt;&gt;&gt; x</td>
</tr>
<tr>
<td>Traceback (most recent call last):</td>
</tr>
<tr>
<td>File &quot;&lt;stdin&gt;&quot;, line 1, in &lt;module&gt;</td>
</tr>
<tr>
<td>NameError: name 'x' is not defined</td>
</tr>
<tr>
<td>&gt;&gt;&gt; module.x</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>&gt;&gt;&gt; help(module)</td>
</tr>
</tbody>
</table>

# This is a comment

```
x = 1+2
x = 3 * x
x
```

"Module data" must be prefixed by module name

Prints **docstring** and module contents

8/28/17
Modules Must be in Working Directory!

Module you want is in this folder
Modules Must be in Working Directory!

Have to navigate to folder **BEFORE** running Python

Module you want is in this folder
Module

- Provides functions, variables
  - **Example**: temp.py
- `import` it into Python shell
  ```python
  >>> import temp
  >>> temp.to_fahrenheit(100)
  212.0
  >>>
  ```

Script

- Behaves like an application
  - **Example**: helloApp.py
- Run it from command line:
  ```bash
  python helloApp.py
  ```

Hello World!
Modules vs. Scripts

Module

- Provides functions, variables
  - Example: temp.py
- import it into Python shell
  >>> import temp
  >>> temp.to_fahrenheit(100)
  212.0
  >>>

Script

- Behaves like an application
  - Example: helloApp.py
- Run it from command line:
  python helloApp.py

Hello World!

Files look the same. Difference is how you use them.
<table>
<thead>
<tr>
<th>module.py</th>
<th>script.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;&quot; A simple module.</td>
<td>&quot;&quot;&quot; A simple script.</td>
</tr>
<tr>
<td>This file shows how modules work</td>
<td>This file shows why we use print</td>
</tr>
<tr>
<td># This is a comment</td>
<td># This is a comment</td>
</tr>
<tr>
<td>x = 1+2</td>
<td>x = 1+2</td>
</tr>
<tr>
<td>x = 3*x</td>
<td>x = 3*x</td>
</tr>
<tr>
<td>x</td>
<td>print(x)</td>
</tr>
</tbody>
</table>
Scripts and Print Statements

module.py

""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x

x

script.py

""" A simple script.

This file shows why we use print
"""

# This is a comment
x = 1+2
x = 3*x

print(x)

Only difference
Script and Print Statements

module.py

- Looks like nothing happens
- Python did the following:
  - Executed the assignments
  - Skipped the last line
  (‘x’ is not a statement)

script.py

- We see something this time!
- Python did the following:
  - Executed the assignments
  - Executed the last line
  (Prints the contents of x)
Scripts and Print Statements

module.py

- Looks like nothing happens
- Python did the following:
  - Executed the assignments
  - Skipped the last line
    (‘x’ is not a statement)

script.py

- We see something this time!
- Python did the following:
  - Executed the assignments
  - Executed the last line
    (Prints the contents of x)

When you run a script, only statements are executed
Next Time: Defining Functions

**Function Call**
- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

**Function Definition**
- Command to **do** the function
- Belongs inside a module
Function Call

- Command to do the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

Function Definition

- Command to do the function
- Belongs inside a module

```
can call as many times as you want
```

But only define function **ONCE**
Next Time: Defining Functions

Function Call

• Command to do the function
• Can put it anywhere
  ▪ In the Python shell
  ▪ Inside another module

Can call as many times as you want

arguments inside ()

Function Definition

• Command to do the function
• Belongs inside a module

But only define function ONCE
Functions and Modules

• Purpose of modules is function definitions
  ▪ Function definitions are written in module file
  ▪ Import the module to call the functions

• Your Python workflow (right now) is

1. Write a function in a module (a .py file)
2. Open up the Terminal/Command Prompt
3. Move to the directory with this file
4. Start Python (type python)
5. Import the module
6. Try out the function